A Scalable Unified System for Seeding Regionalization Queries

Hussah Alrashid
Department of Computer Science and Engineering
University of California, Riverside
Riverside, USA
halra004@ucr.edu

Amr Magdy
Department of Computer Science and Engineering
University of California, Riverside
Riverside, USA
amr@cs.ucr.edu

ABSTRACT

Spatial regionalization is the process of combining a collection of spatial polygons into contiguous regions that satisfy user-defined criteria and objectives. Numerous techniques for spatial regionalization have been proposed in the literature, which employ varying methods for region growing, seeding, optimization and enforce different user-defined constraints and objectives. This paper introduces a scalable unified system for addressing seeding spatial regionalization queries efficiently. The proposed system provides a usable and scalable framework that employs a wide-range of existing spatial regionalization techniques and allows users to submit novel combinations of queries that have not been previously explored. This represents a significant step forward in the field of spatial regionalization as it provides a robust platform for addressing different regionalization queries. The system is mainly composed of three components: query parser, query planner, and query executor. Preliminary evaluations of the system demonstrate its efficacy in efficiently addressing various regionalization queries.

ACM Reference Format:
Hussah Alrashid and Amr Magdy. 2023. A Scalable Unified System for Seeding Regionalization Queries. In *Symposium on Spatial and Temporal Data (SSTD '23), August 23–25, 2023, Calgary, AB, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3609956.3609980

1 INTRODUCTION

Spatial regionalization is a crucial process in numerous domains, aiming to group a collection of spatial polygons (i.e., areas) into spatially contiguous *regions* that minimize a given objective and satisfy user-defined constraints [23]. Each spatial polygon represents a fundamental spatial area, e.g., city block, city, or county. Regionalization has been extensively applied to address various problems across different domains, including epidemic analysis [10], weather temperature classification [24], healthcare resource allocation [31], political districting [38, 39], and metropolitan area delineation [54]. Seeding-based regionalization can be abstracted into two fundamental problems: (1) clustering spatial areas into a predefined number of p regions, as seen in the *p-regions* problem [21], and (2) clustering spatial areas into an arbitrary number of regions, depending on data distribution, as seen in the *max-p-regions* problem [20]. These problems correspond to similar categories in the traditional clustering literature, such as the k-means and DBScan algorithms. However, in this context, clustering is spatially constrained, and the unit is a spatial polygon rather than a d-dimensional point.

There is a multitude of regionalization techniques presented in the literature to address different variations of the *p-regions* and *max-p-regions* problems. These techniques employ a diverse range of region growing, seeding, and optimization tactics while adhering to various user-defined constraints and objectives. Examples include compact regionalization [13, 14, 27, 40–43], multiple constraints regionalization [32], random region regionalization [2, 55], and connected region regionalization [43]. While these approaches differ in certain aspects, they also share some similarities. This unique combination of similarities and differences creates an opportunity to develop a unified framework capable of handling all these techniques using common constructs. The benefits of such a framework are manifold. First, users would be able to submit regionalization queries to a unified system using high-level query languages, eliminating the need to deal with the complexity of implementing various algorithms or optimizations. Second, by supporting a wide range of queries at the system level, this framework would enable system-level optimizations that are not feasible for individual algorithms implemented separately on top of existing computing frameworks, such as Apache Sedona [6]. However, despite the numerous regionalization techniques proposed in the literature, none have yet addressed the need for a unified system for processing regionalization queries. By designing and implementing such a system, researchers can benefit from a more streamlined approach to spatial regionalization, ultimately leading to more efficient and effective solutions in various application domains.

This paper presents *RegioNinja*, a scalable, unified system for addressing regionalization queries. In addition to providing solutions to existing queries, the system enables exploring novel combinations of queries that are not previously investigated, such as *p-regions* queries with average aggregation constraints. It enables users to submit a diverse range of queries tailored to their specific needs, representing a significant advancement in the field of spatial regionalization. *RegioNinja* incorporates a range of techniques, including region-growing algorithms and optimization strategies, to provide efficient and effective solutions for regionalization queries. The utilization of optimization strategies allows the system to deliver high-quality results while minimizing computational costs.

A primary challenge in developing such a unified system lies in identifying the abstract building blocks out of the diverse range

This work is partially supported by the National Science Foundation, USA, under grants IIS-2237348, SES-1831615, and CNS-2031418, and the Google-CAHSI research grant.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SSTD '23, August 23–25, 2023, Calgary, AB, Canada
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0899-2/23/08…$15.00
https://doi.org/10.1145/3609956.3609980

of available regionalization techniques. Abstracting these techniques into a single system that efficiently processes queries is not a straightforward task. This is similar in spirit to identifying SELECT, PROJECT, and JOIN operations for building SQL-based systems. The diversity of existing techniques poses a significant challenge on building blocks that are flexible to support various queries, yet, few enough to be supported in a practical system.

Our framework can function as an independent system with specialized functionality in spatial regionalization or as a major extension for existing spatial query processing engines. This advancement moves the spatial regionalization literature from an era of individual, scattered algorithms to one of system-level support. As a query processing framework, it consists of three components: a query parser, a query planner, and a query executor. The query parser analyzes the input query and extracts elements that are passed to the query planner, which generates execution plans for the executor. Finally, the executor carries out the query plan. Users can pose high-level SQL-like queries, as well as natural language queries (text-based and speech-based) built atop modern AGI platforms, such as GPT-3.5-Turbo APIs. Furthermore, the query executor incorporates several building blocks that enable flexible data preprocessing, region seeding, region growing with various constraints, enclave assignment, and heuristic-based optimization.

One of the prominent features of our system is enabling users to assess the quality of regionalization approximate solutions for the first time in the spatial regionalization literature. This is achieved by enabling statistical inference and generating reference distributions for various statistical tests. This system marks a major advancement in the spatial regionalization literature, offering a robust platform for processing a wide range of regionalization queries.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 outlines basic regionalization definitions. Section 4 formulates the unified seeding regionalization problem. Section 5 provides an overview of the proposed system, and Sections 6- 9 offer a detailed explanation of the system's core components. Finally, Section 10 presents a preliminary evaluation of the system, and Section 11 concludes the paper and discusses future milestones for the system.

2 RELATED WORK

The current literature on spatial regionalization techniques can be broadly classified into two primary categories: regionalization through graph partitioning and regionalization through seeding. These techniques are primarily used to address two types of regionalization problems: regionalization with a predefined number of regions [13, 14, 21, 37, 40–42, 56] and regionalization with an arbitrary number of regions [7, 20, 22, 29, 46, 48, 49, 51–53, 55]. The first type aims to generate p spatially contiguous and homogeneous regions that optionally satisfy certain constraints. In contrast, the second type generates the maximum number of homogeneous regions to determine the appropriate spatial scale of the studied phenomena by enforcing user-defined constraints on the regions to calculate the number of regions automatically. We briefly outline and distinguish our work from the existing literature below.

Graph partitioning techniques. Graph partitioning techniques could be used to generate *p-regions* where the sub-graphs represent the regions and the nodes within the sub-graphs represent the areas. Graph partitioning techniques include SKATER [8] and SKATER-CON [9] to address the *p-regions* problem. In SKATER, edges are removed from a minimum spanning tree (MST) derived from the graph until p sub-graphs are formed. In SKATER-CON, multiple random spanning trees (RST) are generated, and SKATER is applied to each RST. The results of all RSTs are combined to form the final solution. *K-way* graph partitioning [5, 11, 28, 34, 36, 50] aims to generate k sub-graphs with the minimum number of edges connecting the sub-graphs. The graph bisection divides a graph into two sub-graphs of approximately the same number of nodes while minimizing the number of edges between the sub-graphs [19, 25, 26, 33, 35, 44]. Recursively performing a graph bisection on a graph generates p sub-graphs. Node attributed graph partitioning [12, 17, 57] generates k sub-graphs while minimizing the dissimilarity of sub-graphs. Graph partitioning techniques are expensive and cannot solve large datasets, leading to the proposal of seeding techniques, which are the focus of this paper.

Seeding techniques. Seeding techniques grow a region by selecting a seed area and then adding neighboring areas until the region satisfies the constraints. The seeding technique is highly generic and flexible, allowing it to solve the two types of regionalization problems, including the traditional *p-regions* [13, 14, 21, 37, 40, 56] and *max-p-regions* problems [7, 20, 22, 29, 46, 48, 49, 51–53, 55]. Additionally, it is applied to various variations, such as the compact *p-regions* [41, 42] and the compact *max-p-regions* [27] that focus on building regions with a compact shape, and the user-constrained *p-regions* [43] and *max-p-regions* [32] that build regions with one or more user-defined constraints. A few recent techniques address the scalability issue in seeding-based regionalization. A generalized *p-regions* problem called *PRUC* [43] is another seeding problem that solves the *p-regions* problem with a threshold constraint. To optimize the process of growing regions, *PRUC* builds connected regions, i.e., regions where the edges between the areas are maximized. The *EMP* problem [32] is a generalization of the *max-p-regions* problem that generates homogeneous regions while considering multiple constraints, i.e., aggregate functions: MIN, MAX, SUM, COUNT, AVG. There are two techniques that primarily address the scalability issue of the *max-p-regions* problem. The first technique is *SMP* [2], which employs efficient partitioning and region-growing techniques to solve the *max-p-regions* problem. It also presents a novel seed selection technique that aims to reduce the gaps between the regions by selecting the seed for the next region from the neighboring areas for the previously grown region. The second technique [55] introduces a randomized region-growing approach to expedite the traditional *max-p-regions* problem.

Our work falls within the second category of performing seeding-based spatial regionalization; however, it is complementary to all existing work. It proposes a unified system, *RegioNinja*, that answers different combinations of queries, including existing ones and those that have not been studied before such as *p-regions* queries with MIN and AVG constraints. By using a flexible and modular design, our system can handle various types of data and constraints, making it more applicable to real-world problems.

3 PRELIMINARY DEFINITIONS

This section provides definitions for the basic concepts in the regionalization literature.

Definition 1. (Area). An ***area*** a_i is a spatial polygon that is represented by four attributes (i, g_i, s_i, d_i), where i is the area's unique identifier, g_i is the area's geometry represented as an arbitrary spatial polygon, s_i is a spatially extensive attribute, and d_i is a dissimilarity attribute.

An area a_i is an ***articulation area*** if its removal breaks the spatial contiguity of any given set of areas.

The ***area neighbors*** N_{a_i} of any area a_i are the areas that share a boundary (i.e., line or curve) with area a_i.

A ***spatially extensive attribute*** s_i of an area a_i is an attribute whose value is distributed among its smaller k sub-areas $a_{i0}, a_{i1}, ..., a_{ik}$ when the area is divided, and the sum of the attribute values of all sub-areas is equal to the value of the attribute of the original area, i.e., $\sum_{\forall a_{ij} \in a_i} s_{ij} = s_i$. An example of a spatially extensive attribute is the *population* of a county, which is divided over its constituent cities. This is unlike spatially intensive attributes, such as temperature, that are not divided over sub-areas.

Definition 2. (Region). A ***region*** $r_i = \{a_1, a_2, a_3, ..., a_n\}$ is a set of spatially contiguous areas.

The ***region neighbors*** N_{r_i} of any region r_i are the set of areas that do not belong to r_i but have at least one neighbor that belongs to r_i.

A ***spatially contiguous areas*** mean that $\forall a_i, a_j \in R, \exists$ a sequence of areas $\{a_k, ...a_l\}$ such that both a_i, a_k and a_l, a_j are spatial neighbors and every two consecutive areas in the sequence are spatial neighbors.

A ***seed area*** is the first area added to the region (i.e., the area that a region starts growing from). An area a_i is ***assigned*** if it is part of a region and ***unassigned*** if it is not part of any region.

The ***boundary areas*** BA_{r_i} of a region r_i are the areas in r_i that have at least one spatial neighbor that does not belong to r_i.

Definition 3. (User-Defined Constraints). The ***user-defined constraints*** $C = c_i, c_{i+1}, ...c_n$ are conditions that are enforced on the output regions. Each constraint c_i is defined with four attributes $\{f_i, s_i, l_i, u_i\}$ where f_i is one of the aggregate functions: minimum (MIN), maximum (MAX), average (AVG), summation (SUM), and count (COUNT), s_i is the spatially extensive attribute in which the constraint is defined over, l_i is the lower bound of s_i and has a range between $-\infty$ and ∞, u_i is the upper bound of s_i and has a range between $-\infty$ and ∞. So, c_i **is enforced through the inequality** $l_i \leq f_i(s_i) \leq u_i$**.**

Definition 4. (Homogeneous Region). A ***homogeneous region*** r_i is a region where the areas are similar with regards to an attribute, called the dissimilarity attribute d_i, that is associated with each area a_i [20]. The objective is to minimize the dissimilarity between the areas in any region r_i. The dissimilarity for any region r_i is calculated as follows [20]:

$$\sum_{\forall a_{ij}, a_{ik} \in r_i} |d_{ij} - d_{ik}| \tag{1}$$

Definition 5. (Compact Region). A ***compact region*** is a region that is grown with the objective of maximizing its compactness in terms of its shape [27, 40–42]. The compactness of any region r_i is calculated as follows:

$$\frac{S_{r_i}}{2\ \pi\ MI_{r_i}} \tag{2}$$

Where S_{r_i} is the area of region r_i and MI_{r_i} is the second moment of inertia of region r_i.

Definition 6. (Connected Region). A ***connected region*** is a region where the areas are strongly connected with each other. The concept was first introduced in [43] as an optimization to alleviate the cost of growing homogeneous regions. The connectivity between an area a_i and a region r_i is defined as the number of neighbors of a_i that belong to r_i.

Definition 7. (Random Region). A ***random region*** is a region where the areas are added randomly when growing the region. Similar to connected regions, random regions is proposed in [55] as an optimization technique to build regions efficiently.

Definition 8. (Gapless Regions). ***Gapless regions*** are regions that are grown next to each other to minimize the space between them [2]. The seed area for the first region r_1 is picked at random. For the following regions, the seed area is picked from the region neighbors of the previously grown regions. For region r_{i+1}, the seed area is picked from the region neighbors for the previously grown region r_i. If all the region neighbors of r_i are assigned, then the seed area is picked from the region neighbors of r_{i-1} and so on.

Definition 9. (Scattered Seeds). ***Scattered seeds*** are seed areas that are as far away as possible from each other. The distance between two areas a_i and a_j is represented as $dist(a_i, a_j)$ and is defined as the Euclidean distance between the centroids of the two areas. The goal is to maximize the minimum Euclidean distance between the centroids of all pairs of areas in S. The minimum Euclidean distance of a set of areas is defined as follows:

$$min_{a_i \in S \wedge a_j \in S \wedge i \neq j}\ \ dist(a_i, a_j) \tag{3}$$

4 UNIFIED SEEDING REGIONALIZATION PROBLEM

This section provides a formal definition for the unified seeding regionalization problem that abstracts the majority of existing problems in the literature.

Input: (1) A set of n areas; $A = \{a_1, a_2, a_3, ..., a_n\}$. (2) Number of regions p; which could be a positive integer or a flag p_{MAX} indicating the maximum number of regions. (3) Optional set of user-defined constraints; $C = \{c_1, c_2, ..., c_m\}$. If not provided, the number of regions p must be a positive integer. (4) An objective function H computed on a dissimilarity attribute d_i of each area $a_i \in A$.

Output: A set of regions $R = \{r_1, r_2, ..., r_p\}$ of size p, where each region r_i is a non-empty set of spatially continuous areas satisfying the below constraints and objectives.

Constraints:

- $1 \leq p \leq n$
- $|r_i| \geq 1, \quad \forall r_i \in R$
- $r_i \cap r_j = \emptyset, \quad \forall r_i, r_j \in R \wedge\ i \neq j$
- r_i satisfies all the constraints in $C, \quad \forall r_i \in R$

Objectives:

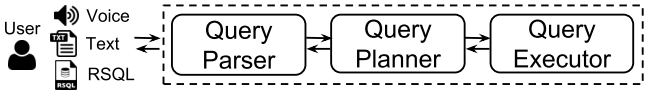


Figure 1: System Architecture

- Minimizing the objective function H.

Examples of the objective function are heterogeneous and compact regions as defined in Equation 1 and Equation 2, respectively.

5 SYSTEM OVERVIEW

This section provides an overview of *RegioNinja*. The system is mainly designed to process the unified seeding regionalization query as defined in Section 4. This problem is generic and allows the discovery of both a pre-defined and arbitrary number of regions. In addition, it solves queries with no constraints at all or with a variety of user-defined constraints that are flexible and support the major SQL-inspired aggregate functions as defined in Section 3. Such a generic query signature incorporates the major work on seeding spatial regionalization queries in the existing literature. Moreover, it enables new combinations of queries that are not possible in existing literature, e.g., *p-regions* query with MAX or AVG aggregate constraints.

Figure 1 depicts the *RegioNinja* system architecture that consists of three components: query parser, query planner, and query executor. We give a brief overview of each one below.

Query Parser. The query parser checks if the query follows the specifications of our query language, called RSQL, and identifies the query elements and passes it to the query planner. The details of the query parser are presented in Section 6.

Query Planner. The query planner generates a query plan from the parsed query. It determines which components should be executed and in what order. The details of the query planner are presented in Section 7.

Query Executor. The query executor executes the query plan generated by the query planner. The details of the query executor are presented in Section 8.

RSQL query language uses four clauses that are inspired by SQL language, `SELECT`, `FROM`, `WHERE`, `ORDER BY`, to express eight basic query processing components that are pipelined to form query plans for various queries. The eight components perform six complementary tasks: (1) Checking the **feasibility** of finding a solution given the user constraints. (2) **Selecting seed areas** with different settings. (3-4) **Growing initial regions** and **assigning enclave areas** to them. (5) **Adjusting the regions' structure** at multiple stages to satisfy all the query constraints. (6) Finding the final regions through **heuristic search optimization** for the initial regions. The RSQL clauses are detailed in Section 6, and the basic query processing components are detailed in Section 8.

This system can function independently or as a major extension of existing spatial query processing engines. The following sections detail each of the core components: *query parser*, *query planner*, and *query executor*.

6 QUERY PARSER

This section introduces our query parser. Our main contribution in this component is introducing new convenient ways for users to pose regionalization queries. Specifically, we introduce the first high-level query language, RSQL, that facilitates posing spatial regionalization queries in SQL-like format for the first time in literature. In addition, we provide natural language interfaces, text-based and speech-based interfaces, built on top of the modern AGI models to enable a wider base of users to interact with *RegioNinja*.

We define specifications for our new query language. The query parser checks if the query follows our query language specifications, parses its different parts, and pass it to the query planner to generate the execution plan. Our query language is directly built on the standard SQL query language that is used in many data management systems. This enables easy integration of our building blocks of spatial regionalization queries into existing systems, which leads to further system-level optimizations at later stages of the system development.

The rest of this section describes the query language specifications and capabilities for the proposed system.

6.1 RSQL Query Signature

Our unified query has the following general form, where the optional clauses and constructs are enclosed within square brackets:

```
SELECT REGIONS, [REGIONS.p, REGIONS.HET];
[ORDER BY (HET | CARD) [(ASC | DESC)];]
FROM (dataset_name | path);
WHERE p = (k | p_MAX),
OBJECTIVE (HETEROGENEOUS | COMPACT) ON attribute_name,
[lower_bound (< | <=) (SUM | MIN | MAX | COUNT | AVG)
(< | <=) upper_bound ON attribute_name],
[OPTIMIZATION (RANDOM | CONNECTED)],
[GAPLESS],
[HEURISTIC (MSA | TABU)];
```

The following sections clarify this general form through query examples and description for different clauses and constructs.

6.2 Query Examples

The following are examples of regionalization queries using our query language:

- ```
  SELECT REGIONS;
  ORDER BY HET DESC;
  FROM US_counties;
  WHERE p=14, GAPLESS,
  11,000 < SUM < 20,000 ON population, 500 <= MIN
  ON population,
  OBJECTIVE HETEROGENEOUS ON average_house_price;
  ```

This query generates 14 regions from the US counties. Each region has a total population that is larger than 11K and less than 20K, and the minimum population of each county is 500. The objective is minimizing the overall heterogeneity of regions based on the average house price in the counties and grow them without gaps in between (gapless) as proposed in [2, 3]. Regions are ordered by heterogeneity in a descending order.

- ```
  SELECT REGIONS, REGIONS.p;
  FROM NYC_census_tracts;
  WHERE p=p_MAX,
  ```

```
5000 <= MAX ON population, OBJECTIVE COMPACT,
OPTIMIZATION CONNECTED, HEURISTIC TABU;
```

This query generates the maximum number of compact regions from the NYC census tracts data and regions are optimized to be more connected. Each region must have a maximum of at least 5K of population in each census tract. The heuristic method used in the local search to optimize the initial solution is tabu search.

6.3 Clauses and Constructs

This section details our query language clauses and constructs. All keywords are case insensitive, e.g., `SELECT` is the same as `select`. Keywords are either clause (i.e., operational) keywords or non-clause keywords. Clause keywords include `SELECT`, `ORDER BY`, `FROM`, `WHERE` and they come at the beginning of each clause in the query. Non-clause keywords include `REGIONS`, `REGIONS.p`, `REGIONS.HET`, `p`, p_{MAX} `DESC`, `ASC`, `CARD`, `HET`, `OBJECTIVE`, `OPTIMIZATION`, `HEURISTIC`, `RANDOM`, `CONNECTED`, `HETEROGENEOUS`, `COMPACT`, `TABU`, `MSA`, `GAPLESS`, `MIN`, `MAX`, `AVG`, `SUM`, `COUNT`, `ON`. Each clause in the query must start with an operational keyword, followed by one or more terms, and end with a semicolon. Below are the description of each clause.

`SELECT` clause instructs to output the regions, the heterogeneity, or the number of regions p of a regionalization result. It is followed by three keywords: (1) `REGIONS`, which outputs regions as a list of pairs of integers, each pair is <*area id*, *region id*>, where all areas affiliated with the same region are labeled with the same *region id*. (2) `REGIONS.p`, which outputs the number of regions (integer), and (3) `REGIONS.HET`, which outputs the regions heterogeneity (double). `REGIONS` is required, while the `REGIONS.p` and `REGIONS.HET` are optional.

`ORDER BY` clause is an optional clause that orders the regions by cardinality or heterogeneity in an ascending or descending order. It is followed by two possible keywords: (1) `CARD`, which orders regions by their cardinality, i.e., number of areas in each region, or (2) `HET`, which orders regions by their heterogeneity values. The order, i.e., `ASC` or `DESC`, is optional and the default value is `ASC`.

`FROM` clause specifies the name or path of the dataset that contains the areas to be regionalized. The supported format is Well-Known Text (WTK) format.

`WHERE` clause is used to provide different query parameters. First, it specifies the number of desired regions, `p`, either a positive integer k ranges from 1 to n (total number of input areas), or a flag p_{MAX} to get the maximum number of regions. Second, it specifies the objective function used to optimize the regions using the keyword `OBJECTIVE`. The currently supported objectives are heterogeneity (`HETEROGENEOUS`) and compactness (`COMPACT`). If the objective is set to `HETEROGENEOUS`, then it must be defined over an attribute using the keyword `ON`. Third, it could be used to specify the optimization criteria in growing initial regions, either to optimize runtime (speed up the process) or the spatial contiguity robustness using the keyword `OPTIMIZATION` which takes the values `RANDOM` or `CONNECTED` or a flag `GAPLESS`. Fourth, it can be used to specify the heuristic algorithm to optimize the initial solution after growing the initial regions using the keyword `HEURISTIC` that is followed two keywords: `MSA` or `TABU`. The default value is `MSA`. The `p` and `OBJECTIVE` are required while the `OPTIMIZATION`, `GAPLESS`, and `HEURISTIC` are optional. Finally, it allows flexible user-defined constraints on multiple attributes of the region. The constraints include the following aggregate functions: summation (`SUM`), minimum (`MIN`), maximum (`MAX`), average (`AVG`), and count (`COUNT`). The aggregate functions are defined over an attribute using the keyword `ON`. A lower bound or upper bound or both are set for each aggregate function as shown in the examples. At least one constraint is required when `p`=p_{MAX}.

6.4 Natural Language Interface

Users of *RegioNinja* can also post queries through a natural language interface, both text-based queries and speech-based queries. This is achieved by integrating Artificial Intelligence chabots, such as ChatGPT [1], with our system. Users then can provide the language description and the query parameters in a natural human language in the chatbot prompt and the interface generates a query according to the given language specifications. To have a more human-like interaction, chat-to-voice extensions, such as Talk to ChatGPT by Google Chrome [16], could also be incorporated to allow users to submit voice queries and receive voice responses.

7 QUERY PLANNER

The query planner is an essential part of *RegioNinja*, specifically designed to support arbitrary queries that are expressed in terms of basic constructs. Our current model operates on a One-Plan-Fits-All paradigm, meaning that all queries are subjected to a similar master plan, except for minor reordering in the adjustment stages. The primary difference lies in the parameterization of the different modules within the plan, which allows tailored execution based on the specifics of each query. The core sequence of operations includes: **Feasibility, Seed Selection, Region Growing, Enclaves Assignment, p Adjustment, Extrema Adjustment, Monotonic Adjustment, and Local Heuristic Search**. The query executor carries out these different stages as detailed in the following section.

8 QUERY EXECUTOR

The query executor executes the different stages of the query plan to produce the query results. This section discusses the operations in order of execution: Feasibility, Seed Selection, Region Growing, Enclaves Assignment, p Adjustment, Extrema Adjustment, Monotonic Adjustment, and Local Heuristic Search.

(1) Feasibility. This phase serves as a crucial initial step, as it determines whether the formulation of a solution is possible given the set of constraints. Based on that, it filters out invalid areas that fail to meet these constraints to produce feasible solutions. In the case of the AVG constraint, if the cumulative average of all areas either falls below the lower bound or exceeds the upper bound, the construction of a solution that fulfills the constraints is an impossibility as proven in [32]. Regarding the MIN constraint, a solution cannot be generated if the smallest value within the areas is above the upper bound or if the maximum value does not reach the lower bound. Under these circumstances, no area satisfies the constraint. Otherwise, any areas with minimum values below the lower bound are removed from the areas. A similar approach is taken with the MAX constraint. If the minimum value is above the upper bound, or the maximum value is below the lower bound, it is

Algorithm 1: Seed Selection

```
1  Input : List of areas A, list of constraints C, number of regions
         p, number of iterations m, Boolean scattered.
2  Output: Set of seed areas S.
3  Initialization:
4  S = {};
5  for each a ∈ A do
6      if a satisfies the MIN or MAX constrains in C then
7          S.add(a);
8  S_random = p seed areas selected randomly from S;
9  if scattered == true then
10     S_not seeds = S − S_random;
11     while m ≠ 0 do
12         a_min = area pair with the minimum Euclidean
             distance;
13         a_random = area selected randomly from S_not seeds;
14         if a_random improves the quality of S if replaced with
             one of the areas in a_min then
15             replace a_random with one of the areas in a_min;
16             m = m − 1;
17 return S;
```

impossible to formulate a solution, as no areas meet the constraint. However, if areas are found with a maximum value that exceeds the upper bound, they must be filtered out.

When it comes to the SUM constraint, if the lowest value exceeds the upper bound, then the construction of a valid solution is impossible. Moreover, if the combined sum of all areas is less than the lower bound, it becomes impossible for any region to satisfy the sum constraint. Otherwise, areas with sum values that exceed the upper bound must be excluded to generate a feasible solution. Finally, in the case of the COUNT constraint, if the total count of areas is less than the lower bound, then no solution can be formed.

(2) Seed Selection. Algorithm 1 outlines the seed selection process. First, all the areas that satisfy the MIN or MAX constraint are filtered and saved in a set S (Lines 5-7). After that, if p is not set to p_{MAX}, then p areas are chosen randomly to be kept in S (Line 8). If the seed strategy is scattered, then the areas in S are replaced with the areas that are not in S to ensure that the seeds in S are as far away as possible from each other (Lines 9-16). An area a_{random} that is not in S is selected at random to replace one of the areas in the area pair having the minimum Euclidean distance. a_{random} is replaced with one of the two areas only if it improves the overall distance of the seeds in S. This process is repeated m times to improve the quality of the seeds. The seed selection strategy, i.e., scattered, is determined by the query planner based on the type of query. For *p-regions* queries, the query planner sets the seed selection strategy to scattered by default to ensure that regions have enough room to grow.

(3) Region Growing. The region growing module serves the purpose of developing initial regions, ensuring adherence to the AVG constraint and the given objective. It starts with the seeds produced by the seed selection component. For each seed area, this component selects a neighboring area that not only satisfies the AVG constraint but also minimizes the objective function, which may either follow the heterogeneity model as depicted in Equation 1 or the compactness model as demonstrated in Equation 2. This selection is performed from the region neighbors N_{ri} of the region r_i. The process of adding areas to a region continues until the AVG constraint is met. If a situation arises where the region does not satisfy the AVG constraint and there are no more region neighbors N_{ri}, the region is retained only when the parameter p is not assigned the value p_{MAX}. In this case, the region's AVG is adjusted in the p adjustment phase. In instances where an optimization (random or connected) is specified, the growth of regions will adhere to the optimization rules, and the objective will be optimized during the local heuristic search phase.

(4) Enclaves Assignment Upon completion of the region growing phase, there might be areas that have not been allocated to any region, referred to as enclaves. The enclave assignment module seeks to allocate these areas to regions through a two-step process. The first phase involves assigning enclaves that align with the AVG constraint. All such enclaves that satisfy the AVG constraint are automatically incorporated into a neighboring region, as integrating an area that already satisfies the average does not violate the AVG constraint of the recipient region. The subsequent step addresses the remaining enclaves that fail to satisfy the AVG constraint. This step is conducted by iterating over these enclaves, and for each enclave, neighboring regions are identified. The updated average of a region, computed by considering the enclave area, is then assessed. If the newly calculated average of the region still meets the AVG constraint, the enclave area is added to the region. This procedure continues until all enclaves are allocated to regions. However, if certain enclaves violate the AVG constraint for all neighboring regions, they are eliminated from the solution, only if the parameter p has been set to p_{MAX}. Otherwise, the enclaves are added to the regions randomly even if they do not satisfy the AVG constraint and the regions' AVG will be adjusted in the p Adjustment phase.

(5) p **Adjustment**. This phase is executed when the number of regions, p, is assigned a positive integer value. After the enclave assignment, there may remain incomplete regions that do not yet fulfill the AVG constraint. The p adjustment aims to ensure these incomplete regions satisfy the AVG constraint. For each incomplete region, all neighboring complete regions are identified, and areas are moved from these neighboring complete regions (i.e., donor regions) to the incomplete region (i.e., receiver region). Neighboring regions are determined by examining whether any neighboring areas of the incomplete region's areas belong to another region. Subsequently, boundary areas of the neighboring complete regions are moved to the incomplete region, provided they are not articulation areas. This move may cause donor regions to become incomplete. A subsequent step resolves this issue by identifying all complete regions in a list, then moving areas from complete regions to neighboring incomplete regions in the same manner. Following this step, all regions should satisfy the AVG constraint.

(6) Extrema Adjustment The seed selection identifies the areas that satisfy either the MIN or MAX constraints. Therefore, regions are guaranteed to satisfy only one of those constraints. To ensure that regions satisfy both constraints, the regions are checked one by one. If the region satisfies one of the constrains, it is merged with a neighboring region that satisfies the other constraint.

(7) Monotonic Adjustment To ensure compliance with the SUM and COUNT constraints for each formed region, it may become necessary to move areas among regions. For both the SUM and COUNT constraints, if a region exceeds the upper bound, areas are moved to an adjacent region, providing this does not affect the constraints of the receiving region. Conversely, if a region falls below the lower bound, areas are added from neighboring regions, ensuring this action does not violate the constraints of the donating region. If the SUM or COUNT constraints are still not satisfied after this step, then areas are removed from regions that are above the upper bound or the regions are merged when they are below the lower bound. The merging process repeatedly merges two regions when at least one of them is below the threshold without violating the constrains. Any regions that remain non-compliant post-adjustment are excluded, thus finalizing the solution.

(8) Local Heuristic Search. This component focuses on optimizing the quality of the solution based on the given objective. The basic idea of local search algorithms [30, 55] is moving areas from one region (the donor region) to another neighboring region (the recipient region). An area is considered moveable only if it upholds two conditions: (a) It preserves the spatial continuity of the donor region, and (b) it does not violate the constraints of both regions.

Current techniques basically employ two local search algorithms. The first is the Tabu Search [30]. The aim of the algorithm is to move areas among neighboring regions while adhering to user-defined constraints. The algorithm starts with an initial solution and proceeds iteratively towards the best possible solution. To escape a local optimum solution, the algorithm allows moves towards solutions with a worse objective function value. Tabu Search maintains a tabu list that saves moves executed during the search to prevent reverse movements that may trigger cycles. If a move results in a solution better, the algorithm selects that move, even if it is in the tabu list. The search terminates when no further advantageous moves can be made within a specified limit of steps, and the best solution found so far is returned as the final result.

The second algorithm is the Modified Simulated Annealing (MSA) [2, 3, 55]. It starts by generating a set of movable areas. Then, it randomly selects an area from the list and shifts it to another region if the move enhances the objective function value of the existing solution. If not, the move is accepted with a probability calculated using the Boltzmann's equation: $e^{-\Delta H/TM}$ where $-\Delta H$ represents the value of the objective function variation on both donor and recipient regions, and *TM* represents the temperature. The temperature *TM* is decreased at each iteration at a fixed cooling rate *PH* until *TM* reaches a predefined value. In contrast to the original simulated annealing algorithm [20], the MSA algorithm introduces two efficient techniques to expedite the optimization process. First, it reuses the set of moveable areas until it becomes empty instead of recalculating the set each time an area is moved [55]. Second, it uses a tabu list with a specific size, i.e., *LI*, to prevent cycles. Finally, it employs Tarjan's algorithm [47] to find all the articulation areas in a single graph traversal to identify the moveable areas [2, 3].

9 SPATIAL STATISTICAL INFERENCE

All current regionalization techniques employ heuristics that produce approximate solutions without providing a guarantee of their quality. Experts within the respective field need to perform statistical inference to evaluate the quality of resultant solutions. In order to conduct statistical inference, one would need to generate a reference distribution comprising sample solutions that can be used for comparative analysis against the evaluated solution. These sample solutions, known as the reference distribution, must be similar to the evaluated solution in terms of the number of regions and the cardinality of each region (i.e., the number of areas within each region) to ensure the provision of significant statistical evidence.

To incorporate spatial statistical inference into *RegioNinja*, a keyword (i.e., `REGIONS.DISTRIBUTION`) is included in the `SELECT` clause to output a reference distribution of a user-defined sample size, with a default sample size of 100 random solutions. In addition, the Monotonic Adjustment component of the query executor is extended to include an operation to adjust the cardinality of the merged region. After each merge iteration, the smallest cardinality is checked against all the merged regions. If the size of all the regions is larger than the smallest cardinality, then areas are moved from the smallest region to neighboring regions until its size matches the desired cardinality. This process is repeated at each iteration for each cardinality value, as detailed in our work [4].

Table 1: Evaluation Datasets Description

Name	No. of Areas	States
10K	$\approx 10 \times 10^3$	CA, NV, and AZ.
20K	$\approx 20 \times 10^3$	10K, OR, WA, ID, UT, MT, WY, CO, NM, OK, KS, NE, SD, and ND.
30K	$\approx 30 \times 10^3$	20K, TX, LA, AR, MO, and IA.
40K	$\approx 40 \times 10^3$	30K, MN, MS, AL, TN, KY, IL, and WI.
50K	$\approx 50 \times 10^3$	40K, GA, IN, MI, OH, and WV.
60K	$\approx 60 \times 10^3$	50K, FL, SC, NC, VA, and MD.
70K	$\approx 70 \times 10^3$	60K, PA, NY, NJ, and DE.

10 SYSTEM EVALUATION

This section provides a preliminary evaluation for *RegioNinja*. We evaluate two aspects: system performance and system usability.

10.1 System Performance

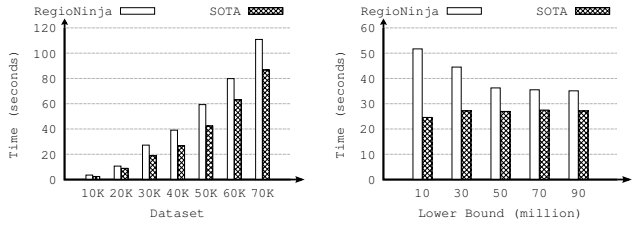
We currently support two families of queries representing the two major categories in the existing literature: (1) *p-regions* query and its variations, and (2) *max-p-regions* query and its variations. We use three **performance measures**: *runtime* to measure the system's scalability, *number of regions p* and *heterogeneity H* to measure the quality of the generated solutions.

Our **evaluation datasets** are all subsets of the census tracts TIGER/Line shape files of the US states [15]. Areas in each dataset represent the census tracts for certain US states. The details of the datasets are presented in Table 1. In our experiment, the spatially extensive attribute is defined over the *aland* attribute that represents the land area. The dissimilarity attribute is defined over the *awater* attribute that represents the water area. All the experiments are based on Java 14 implementation and run on Ubuntu 16.04 with a quad-core 3.5GHz processor and 128GB of memory.

10.1.1 ***Max-p-regions Queries***. The *max-p-regions* query of the proposed system is denoted as ***RegioNinja*** and is evaluated

Table 2: Evaluation parameters for *max-p-regions* queries

Parameter	Values
DS	10K, 20K, 30K, **40K**, 50K, 60K, 70K
l	10, 30, **50**, 70, 90 ($\times 10^6$)



(a) Runtime varying *DS* values **(b) Runtime varying *l* values**

Figure 2: Impact of the parameters on the runtime of the *max-p-regions* queries

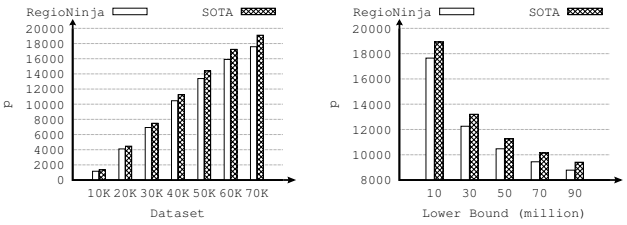
against the state-of-the-art *max-p-regions* problem, which is denoted as ***SOTA*** [55]. The system's *max-p-regions query*, *RegioNinja*, have the `OPTIMIZATION` set to `RANDOM` and the `OBJECTIVE` set to `HETEROGENEOUS`. It has one constraint, which is SUM, with an upper bound set to ∞ and varying lower bound (l) values specified in Table 2. We evaluate the performance of the *max-p-regions* queries with the following parameters:

- *DS*: dataset size, which represents the number of areas in the dataset.
- *l*: lower bound of the SUM constraint.

Table 2 presents the parameters' values used in this experiment. The default value for each parameter is indicated in bold. The local heuristic search parameters: tabu list length (*LI*), the temperature of computing the Boltzmann's probability (*TM*), and its cooling rate (*PH*), are set to 100, 1, and 0.9, respectively, as proposed in [20].

Impact of the parameters' values on the runtime. Figure 2 shows the impact of changing the *DS* and *l* values on the runtime for *RegioNinja* and *SOTA*. Increasing the *DS* value increases the runtime for all alternatives as shown in Figure 2a as there are more areas to process in general. Increasing the *l* value decreases the runtime of *RegioNinja* while the runtime of *SOTA* is almost consistent with different values. The merging process in the SUM adjustment merges each region only once in each iteration and updates the region neighbors each time it is merged. When the lower bound value *l* is small, regions are more likely to satisfy the constraint in early iterations, which lead to leaving some regions not merged which in turn increases the number of iterations. Having more iterations increases the runtime since the number of updates for the neighborhood relationships between regions will increase. This explains why the runtime of *RegioNinja* decreases as we increase the *l* value. *RegioNinja* is slightly slower than *SOTA* due to its modular design. The query have to go through the feasibility, seed selection, and region growing components (which filters out each single area as a region) before reaching the SUM adjustment. Overall, *RegioNinja* is still efficient enough to handle large-sized datasets while hiding all the complications from end-users and enabling future system-level optimizations.

Impact of the parameters' values on p and the Heterogeneity. Increasing the *DS* value increases the *p* for all alternatives as there



(a) *p* varying *DS* values **(b) *p* varying *l* values**

Figure 3: Impact of the parameters on *p* of the *max-p-regions* queries

Table 3: Evaluation parameters' values for *p-regions* queries

Parameter	Values
DS	10K, 20K, 30K, **40K**, 50K, 60K, 70K
p	0.01%, 0.02%, **0.03%**, 0.04%, 0.05% (*of DS*)
m	0.0001%, 0.0002%, **0.0003%**, 0.0004%, 0.0005% (*of DS*)

are more input areas as shown in Figure 3a. On the other hand, Figure 3b shows that increasing the *l* value decreases the number of regions. This is due to the fact that more areas will be needed in the region to reach the lower bound when the *l* value increases. *RegioNinja* generates slightly less regions than *SOTA* since the merging process reduces the granularity of the areas in the SUM adjustment. At each iteration, two regions (initially two areas in the first iteration) are merged. After that, merging two regions leads to adding more than one area to the region which results in less regions. Whereas in *SOTA*, areas are added one by one to the region until it reaches the lower bound.

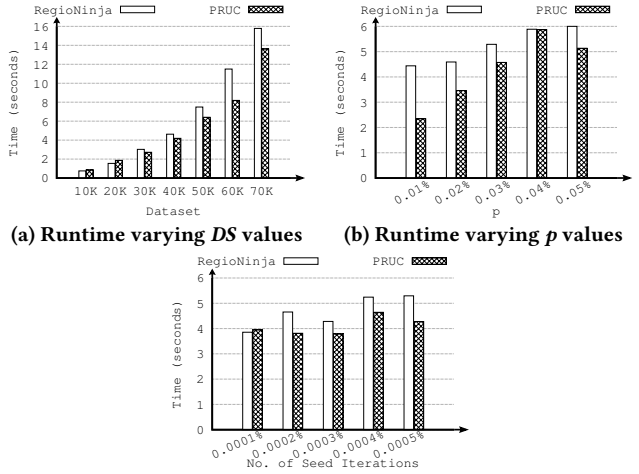
The *p* directly affects the heterogeneity. *RegioNinja* has a slightly higher heterogeneity value than *SOTA* since it generates less regions. This is because having a larger number of regions means having more area pairs in each region that contribute to the heterogeneity. Overall, *RegioNinja* is able to generate solutions that are comparable to those produced by the existing techniques efficiently while offering a convenient way for users to submit queries.

10.1.2 ***P-regions Queries.*** The *p-regions* query of the proposed system is denoted as ***RegioNinja*** and is evaluated against the generalized *p-regions* problem PRUC, denoted as ***PRUC*** [43]. The system's *p-region* query, *RegioNinja*, have the `OPTIMIZATION` set to `RANDOM` and the `OBJECTIVE` set to `HETEROGENEOUS`. We evaluate the performance of the *p-regions* queries with the following parameters:

- *DS*: dataset size, which represents the number of areas in the dataset.
- *p*: number of regions, which is a percentage of the dataset size *DS*.
- *m*: number of seed iterations, which is a percentage of the dataset size *DS*.

Table 3 presents the parameters' values used in this experiment. The default value for each parameter is indicated in bold.

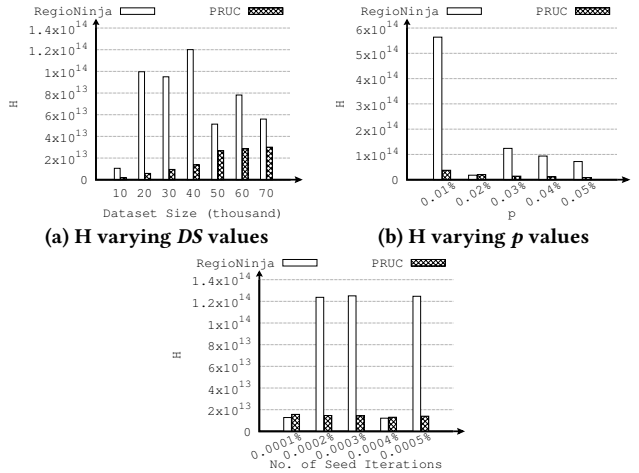
Impact of the parameters' values on the runtime. The impact of changing the *DS*, *p*, and *m* values on the runtime for *RegioNinja* and *PRUC* are shown in Figure 4. Increasing the *DS* value increases the runtime for all alternatives as shown in Figure 4a as there are more areas to process in general. On the other hand, Figure 4b and Figure 4c shows that the run time is consistent with different values



(a) Runtime varying *DS* values (b) Runtime varying *p* values

(c) Runtime varying *m* values

Figure 4: Impact of the parameters on the runtime of the *p-regions* queries



(a) H varying *DS* values (b) H varying *p* values

(c) H varying *m* values

Figure 5: Impact of the parameters on the heterogeneity of the *p-regions* queries

(only differs by a few seconds). The performance of *RegioNinja* and *PRUC* is the same, which proves the ability of our system to generate solutions in a very efficient manner while providing a usable way to submit queries.

Impact of the parameters' values on the Heterogeneity. Figure 5 shows that there is no correlation between the *DS*, *p*, and *m* values on the heterogeneity for *RegioNinja* and *PRUC*. The reason for this is that the regions are grown at random in both queries, which gives very different results each time. The heterogeneity of the *RegioNinja* is very close to that of the *PRUC*, which proves the ability of our system to provide a usable way to submit queries and generate solutions with a quality similar to the original query.

10.2 System Usability

To run the state-of-the-art *max-p-regions* [55], one would have to write a lengthy code (1855 lines), and 2209 lines of code to run PRUC [43]. Our proposed system allows users to execute the *max-p-regions* query with only a few lines as follows:

```
SELECT REGIONS;
FROM census_tracts;
WHERE p=p_MAX,
OPTIMIZATION RANDOM, OBJECTIVE HETEROGENEOUS ON awater,
SUM >= 90,000,000 ON aland;
```

This applies to all the other existing queries. *RegioNinja* provides a framework where users can submit different queries within seconds instead of having to implement each query from scratch.

11 FUTURE DIRECTIONS AND CONCLUSIONS

This paper introduces an innovative and unified system for spatial regionalization, combining a variety of techniques to achieve effective and efficient results. The system serves as a practical and scalable framework that not only accommodates traditional seeding regionalization queries, but also introduces novel combinations of queries unexplored to date. The system engages a variety of techniques, encompassing region-growing algorithms and optimization strategies that can be tailored to cater to specific user requirements. The adoption of optimization strategies empowers the system to yield high-quality results while concurrently minimizing computational expenses. This breakthrough represents a significant step forward in the field of spatial regionalization, offering a robust and versatile platform for addressing a broad spectrum of regionalization queries. Preliminary evaluations of the system proves its effectiveness in addressing regionalization queries efficiently.

The presented system paves the way for future significant enhancements in the spatial regionalization domain. Such enhancements range from advancing existing components of the system to adding brand-new components to advance the existing literature.

For instance, future iterations should accommodate a broader range of flexible user-defined constraints that cater to real-world applications, e.g., variance and standard deviation constraints. Also, the system could support more objective functions within a generalized scheme, yet continue to prioritize key functions. The system could also be made more user-friendly to facilitate handling various dataset formats. Further, the optimization and adjustment strategies could be improved to yield higher-quality solutions across diverse query combinations. Thus, different opportunities for system improvements and optimizations could significantly elevate the efficiency and quality of regionalization queries.

The system is also designed to incorporate revolutionary components that could fundamentally transform the way users approach spatial regionalization queries. The inclusion of the natural language interface component is particularly significant. This component could be significantly improved by enhancing the expressiveness of queries and offering customized and personalized user experiences. This could be achieved through deeper and more advanced integration with Large Language Models (LLMs) and relevant systems. For example, Microsoft's DeepSpeed-Chat [45], which provides a simple and cost-effective means of training ChatGPT-like models using reinforcement learning techniques based on human interaction feedback, could be considered. With DeepSpeed-Chat, a 13-billion parameter model can be trained on a single GPU, or on

Microsoft's Azure Cloud platform, at a low cost of just $300 and within a relatively short timeframe, typically less than 12 hours. Another potential avenue to explore is Databricks Dolly [18], an open system that offers additional avenues for developing natural language interfaces. The integration with such systems could dramatically enhance the user experience, marking a significant advancement in spatial regionalization technology.

REFERENCES

[1] Open AI. 2023. Introducing ChatGPT. https://openai.com/blog/chatgpt.
[2] Hussah Alrashid, Yongyi Liu, and Amr Magdy. 2022. SMP: Scalable Max-P Regionalization. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, USA, 1–4.
[3] Hussah Alrashid, Yongyi Liu, and Amr Magdy. 2023. PAGE: Parallel Scalable Regionalization Framework. *Under minor revision in TSAS* (2023), 1–27.
[4] Hussah Alrashid, Amr Magdy, and Sergio Rey. 2023. Statistical Inference for Spatial Regionalization. In *Under submission to SIGSPATIAL*. Association for Computing Machinery, New York, NY, USA, 1–10. https://drive.google.com/file/d/1m1C7IYhK6155U0Idsqa4YCBWqPHB22Lf/view.
[5] Konstantin Andreev and Harald Racke. 2006. Balanced Graph Partitioning. *Theoretical Computer Science* 39, 6 (2006), 929–939.
[6] Apache. 2023. Apache Sedona. https://sedona.apache.org/latest-snapshot/.
[7] Daniel Arribas-Bel and Charles R Schmidt. 2013. Self-Organizing Maps and the US Urban Spatial Structure. *EPB* 40 (2013), 362–371.
[8] Renato M Assunção, Marcos Corrêa Neves, Gilberto Câmara, and Corina da Costa Freitas. 2006. Efficient Regionalization Techniques for Socio-economic Geographical Units Using Minimum Spanning Trees. *IJGIS* 20 (2006), 797–811.
[9] Orhun Aydin, Mark V Janikas, Renato Assunção, and Ting-Hwan Lee. 2018. SKATER-CON: Unsupervised Regionalization via Stochastic Tree Partitioning Within a Consensus Framework Using Random Spanning Trees. In *ACMGeoAI*. Association for Computing Machinery, New York, NY, USA, 33–42.
[10] Roberto Benedetti, Federica Piersimoni, Giacomo Pignataro, and Francesco Vidoli. 2020. The Identification of Spatially Constrained Homogeneous Clusters of Covid-19 Transmission in Italy. *RSPP* 12 (2020), 1169–1187.
[11] Una Benlic and Jin-Kao Hao. 2011. An Effective Multilevel Tabu Search Approach for Balanced Graph Partitioning. *Operations Research* 38, 7 (2011), 1066–1075.
[12] Daniel Bereznyi, Ahmad Qutbuddin, YoungGu Her, and KwangSoo Yang. 2020. Node-attributed Spatial Graph Partitioning. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, USA, 58–67.
[13] Subhodip Biswas, Fanglan Chen, Zhiqian Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2020. Incorporating Domain Knowledge into Memetic Algorithms for Solving Spatial Optimization Problems. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, USA, 25–35.
[14] Subhodip Biswas and et. al. 2019. REGAL: A Regionalization Framework for School Boundaries. In *SIGSPATIAL*. Association for Computing Machinery, New York, NY, USA, 544–547.
[15] U.S. Census Bureau. 2019. TIGER/Line Shapefile, 2016, Series Information for the Current Census Tract State-based Shapefile. https://catalog.data.gov/dataset/tiger-line-shapefile-2016-series-information-for-the-current-census-tract-state-based-shapefile.
[16] Google Chrome. 2023. Talk to ChatGPT. https://github.com/C-Nedelcu/talk-to-chatgpt.
[17] David Combe, Christine Largeron, Elöd Egyed-Zsigmond, and Mathias Géry. 2012. Combining Relations and Text in Scientific Network Clustering. In *ASONAM*. IEEE Computer Society, Washington, D.C., USA, 1248–1253.
[18] Mike Conover and et. al. 2023. Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM. https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm.
[19] Daniel Delling, Daniel Fleischman, Andrew V Goldberg, Ilya Razenshteyn, and Renato F Werneck. 2015. An Exact Combinatorial Algorithm for Minimum Graph Bisection. *Mathematical Programming* 153, 2 (2015), 417–458.
[20] Juan C Duque, Luc Anselin, and Sergio J Rey. 2012. The Max-P-Regions Problem. *JRS* 52 (2012), 397–419.
[21] Juan C Duque, Richard L Church, and Richard S Middleton. 2011. The P-Regions Problem. *Geographical Analysis* 43 (2011), 104–126.
[22] Juan C Duque, Jorge E Patino, Luis A Ruiz, and Josep E Pardo-Pascual. 2015. Measuring Intra-urban Poverty Using Land Cover and Texture Metrics Derived from Remote Sensing Data. *LUP* 135 (2015), 11–21.
[23] Juan Carlos Duque, Raúl Ramos, and Jordi Suriñach. 2007. Supervised Regionalization Methods: A Survey. *IRSR* 30 (2007), 195–220.
[24] Ahmed El Kenawy, Juan I López-Moreno, and Sergio M Vicente-Serrano. 2013. Summer Temperature Extremes in Northeastern Spain: Spatial Regionalization and Links to Atmospheric Circulation (1960–2006). *TAC* 113 (2013), 387–405.
[25] Uriel Feige and Robert Krauthgamer. 2002. A Polylogarithmic Approximation of the Minimum Bisection. *SICOM* 31, 4 (2002), 1090–1118.
[26] Ariel Felner. 2005. Finding Optimal Solutions to the Graph Partitioning Problem with Heuristic Search. *AMAI* 45, 3 (2005), 293–322.
[27] Xin Feng, Sergio Rey, and Ran Wei. 2022. The max-p-compact-regions Problem. *Transactions in GIS* 26, 2 (2022), 717–734.
[28] Thomas Feo, Olivier Goldschmidt, and Mallek Khellaf. 1992. One-Half Approximation Algorithms for the k-Partition Problem. *Operations Research* 40 (1992), S170–S173.
[29] David C Folch and Seth E Spielman. 2014. Identifying Regions Based On Flexible User-defined Constraints. *IJGIS* 28 (2014), 164–184.
[30] Fred Glover. 1989. Tabu Search—Part I. *ORSA Journal on Computing* 1 (1989), 190–206.
[31] Jeremiah Hurley. 2004. Regionalization and the Allocation of Healthcare Resources to Meet Population Health Needs. *Healthcare Papers* 5 (2004), 34–39.
[32] Yunfan Kang and Amr Magdy. 2022. EMP: Max-P Regionalization with Enriched Constraints. In *ICDE*. IEEE, 1914–1926.
[33] David R Karger. 1993. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm.. In *SODA*. Society for Industrial and Applied Mathematics, USA, 21–30.
[34] George Karypis and Vipin Kumar. 1998. Multilevel K-Way Partitioning Scheme for Irregular Graphs. *J. Parallel and Distrib. Comput.* 48, 1 (1998), 96–129.
[35] Brian W Kernighan and Shen Lin. 1970. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal* 49, 2 (1970), 291–307.
[36] MS Khan and KF Li. 1995. Fast Graph Partitioning Algorithms. In *PACRIM*. IEEE Computer Society, Washington, D.C., USA, 337–342.
[37] Hyun Kim, Yongwan Chun, and Kamyoung Kim. 2015. Delimitation of Functional Regions Using a P-Regions Problem Approach. *IRSR* 38 (2015), 235–263.
[38] Myung Kim and Ningchuan Xiao. 2017. Contiguity-based Optimization Models for Political Redistricting Problems. *IJAGR* 8, 4 (2017), 1–18.
[39] Yunfeng Kong, Yanfang Zhu, and Yujing Wang. 2019. A Center-based Modeling Approach to Solve the Districting Problem. *IJGIS* 33, 2 (2019), 368–384.
[40] Jason Laura, Wenwen Li, Sergio J Rey, and Luc Anselin. 2015. Parallelization of a Regionalization Heuristic in Distributed Computing Platforms–a Case Study of Parallel-P-Compact-Regions Problem. *IJGIS* 29 (2015), 536–555.
[41] Wenwen Li, Richard L Church, and Michael F Goodchild. 2014. An Extendable Heuristic Framework to Solve the P-Compact-Regions Problem for Urban Economic Modeling. *CEUS* 43 (2014), 1–13.
[42] Wenwen Li, Richard L Church, and Michael F Goodchild. 2014. The p-Compact-Regions Problem. *Geographical Analysis* 46 (2014), 250–273.
[43] Yongyi Liu, Ahmed R. Mahmood, Amr Magdy, and Sergio Rey. 2022. PRUC: P-Regions with User-Defined Constraint. In *VLDB*. 491–503.
[44] Joe Marks, Wheeler Ruml, Stuart Shieber, and J Thomas Ngo. 1998. A Seed-Growth Heuristic for Graph Bisection. *Proceedings of Algorithms and Experiments' 98* (1998), 76–87.
[45] Microsoft. 2023. DeepSpeed Chat: Easy, Fast and Affordable RLHF Training of ChatGPT-like Models at All Scales. https://github.com/microsoft/DeepSpeed/tree/master/blogs/deepspeed-chat.
[46] Jorge E Patino, Juan C Duque, Josep E Pardo-Pascual, and Luis A Ruiz. 2014. Using Remote Sensing to Assess the Relationship Between Crime and the Urban Layout. *Applied Geography* 55 (2014), 48–60.
[47] Tarjan R. 1971. Depth-first Search and Linear Graph Algorithms. *SICOM* 1 (1971), 114–121.
[48] Sergio J Rey, Luc Anselin, David C Folch, Daniel Arribas-Bel, Myrna L Sastré Gutiérrez, and Lindsey Interlante. 2011. Measuring Spatial Dynamics in Metropolitan Areas. *EDQ* 25 (2011), 54–64.
[49] Sergio J Rey and Myrna L Sastré-Gutiérrez. 2010. Interregional Inequality Dynamics in Mexico. *SEA* 5 (2010), 277–298.
[50] Kirk Schloegel, George Karypis, and Vipin Kumar. 2000. Parallel Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *European Conference on Parallel Processing*. Springer, Berlin, Heidelberg, 296–310.
[51] Bing She, Juan C Duque, and Xinyue Ye. 2017. The Network-Max-P-Regions Model. *IJGIS* 31 (2017), 962–981.
[52] V. Sindhu. 2018. *Exploring Parallel Efficiency and Synergy for Max-P Region Problem Using Python*. Master's thesis. Georgia State University.
[53] Seth E Spielman and David C Folch. 2015. Reducing Uncertainty in the American Community Survey through Data-driven Regionalization. *PloS ONE* 10 (2015), e0115626.
[54] Daoqin Tong and David A Plane. 2014. A New Spatial Optimization Perspective on the Delineation of Metropolitan and Micropolitan Statistical Areas. *Geographical Analysis* 46, 3 (2014), 230–249.
[55] Ran Wei, Sergio Rey, and Elijah Knaap. 2020. Efficient Regionalization for Spatially Explicit Neighborhood Delineation. *IJGIS* 35 (2020), 1–17.
[56] Xinyue Ye, Bing She, and Samuel Benya. 2018. Exploring Regionalization in the Network Urban Space. *JGSA* 2 (2018), 4.
[57] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph Clustering Based on Structural/Attribute Similarities. *PVLDB* 2 (2009), 718–729.